MACHINE LEARNING

what it is and how to get started

Auralee Edelen
May 4, 2017

MACHINE LEARNING

what it is and how to get started

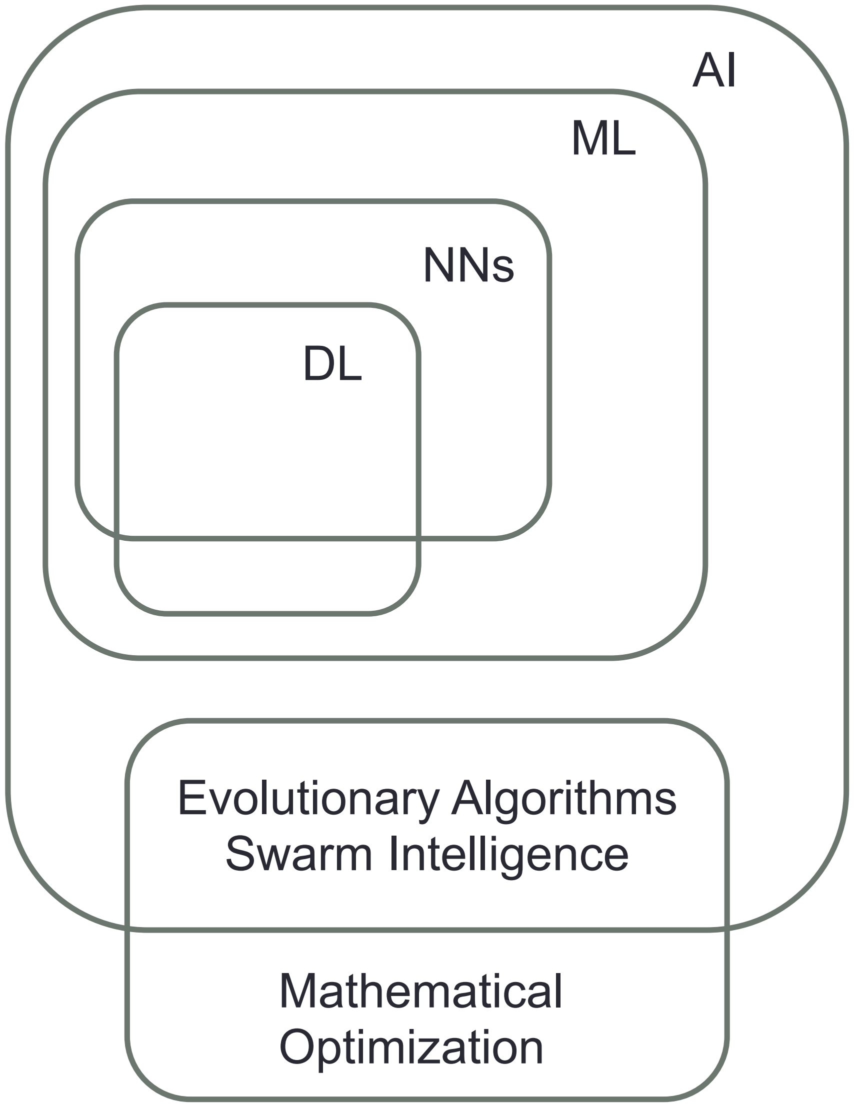
Auralee Edelen
May 4, 2017

Caveats!

— 20 minutes → very high-level overview
— heavy bias toward neural networks

Field Taxonomy (as of now…)

- Artificial Intelligence (AI)
  - *Field of getting machines to exhibit aspects of human intelligence, esp. knowledge, learning, planning, reasoning, perception*
  - Narrow AI: focused on a task or similar set of tasks
  - General AI: human-equivalent or greater performance on any task
- Machine Learning (ML)
  - *Field of getting machines to complete tasks without being explicitly programmed*
  - Common tasks: Regression, Classification, Clustering, Dimensionality Reduction
- Neural Networks (NNs)
  - *A set of tools within ML that uses a many connected processing units*
  - Many kinds: feed-forward, recurrent, adversarial, self-organizing maps
  - Very popular right now (somewhere at the top of the hype cycle…)
- Deep Learning (DL)
  - *Learning hierarchical representations*
  - Right now, largely synonymous with methods based on deep (many-layered) NNs



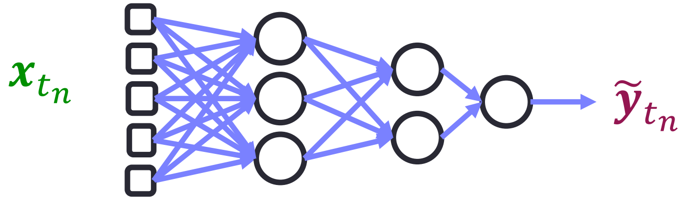
Note that these definitions are not rigid: there is a lot of fluidity in the field at the moment!

Basic Learning Paradigms

Example: Regression using a NN

Data set of **input** and **output** pairs:

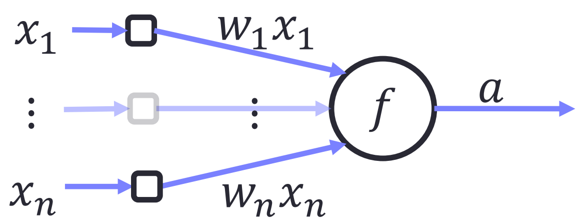
$$\left.\begin{array}{c} x_1 \\ \vdots \\ x_n \end{array}\right\} \quad \begin{array}{cc} \boldsymbol{x}_{t_1} & \boldsymbol{y}_{t_1} \\ \vdots & \vdots \\ \boldsymbol{x}_{t_n} & \boldsymbol{y}_{t_n} \end{array}$$



Want to find approximate map:

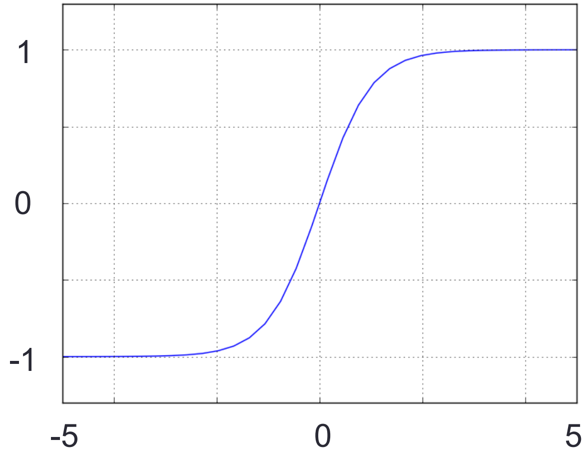
$$\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{y}$$

ANN Basic Structure

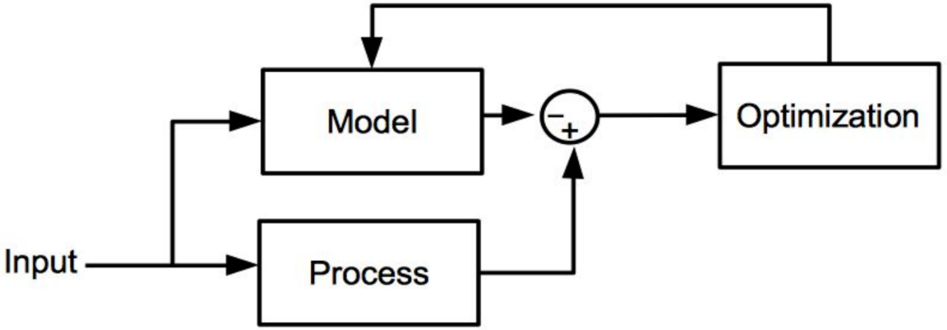


$$f\left(\sum_n w_n x_n + b\right) = a$$

e.g. $f(z) = \frac{2}{(1+e^{-2z})} - 1$



Model Learning



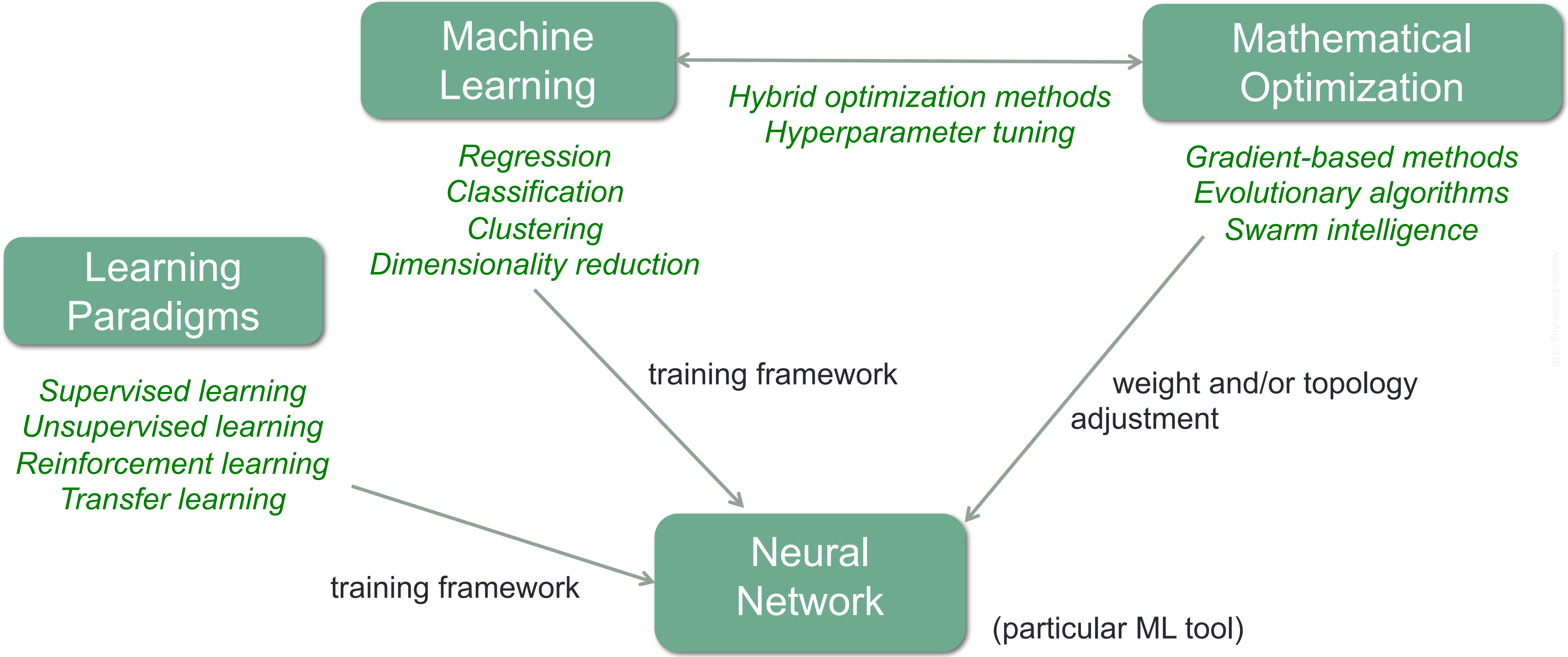
Example:

$$C(w,b) = \frac{1}{2t_n}\left[\sum_{t_n}(y_{t_n} - \tilde{y}_{t_n})^2\right]$$

$$w_k \longrightarrow w'_k = w_k - \alpha \frac{\partial C}{\partial w_k}$$

$$b_k \longrightarrow b'_k = b_k - \alpha \frac{\partial C}{\partial b_k}$$

Example of how this all fits together for NNs



ML Software and Related Libraries

- **Theano** – library for fast numerical computation (graph-based, automatic differentiation, python)
- **Tensor Flow** – library for fast numerical computation (graph-based, automatic differentiation, mostly python but some support for Java, C, Go)
- **Torch** – machine learning and scientific computing framework (Lua)
- **Scikit-learn** – library for general machine learning (python)
- **Caffe** – neural network framework (python interface, written in C++, popular in HEP, large library of pre-trained models)
- **Chainer** – neural network framework (python)
- **Lasagne** – neural network library over Theano (python)
- **Keras** – neural network library over Theano/Tensor Flow (python, also higher-level than Lasagne)
- **MATLAB** neural network toolbox

- Bare bones example of how things are structured in Theano and Lasagne

- Easy to set up mechanically

→ *much of the difficulty in using NNs comes with the training process and defining the initial problem correctly*

```python
import theano
import theano.tensor as T
import lasagne

lin = lasagne.layers.InputLayer(shape=(None, 500), input_var=input_var)

l1 = lasagne.layers.DenseLayer(lin,
    num_units = 100, nonlinearity=lasagne.nonlinearities.tanh,
    W=lasagne.init.GlorotUniform(gain=1),b=lasagne.init.Normal(std=0.001, mean=0.0))

l2 = lasagne.layers.DenseLayer(l1,
    num_units = 70, nonlinearity=lasagne.nonlinearities.tanh,
    W=lasagne.init.GlorotUniform(gain=1),b=lasagne.init.Normal(std=0.001, mean=0.0))

l3 = lasagne.layers.DenseLayer(l2,
    num_units = 10, nonlinearity=lasagne.nonlinearities.tanh,
    W=lasagne.init.GlorotUniform(gain=1),b=lasagne.init.Normal(std=0.001, mean=0.0))

out = lasagne.layers.DenseLayer(l3,
    num_units = 1, nonlinearity=lasagne.nonlinearities.linear,
    W=lasagne.init.GlorotUniform(gain=1),b=lasagne.init.Normal(std=0.001, mean=0.0))

input_var = T.matrix('inputs', dtype='float32')
target_var = T.matrix('targets',dtype='float32')

prediction = lasagne.layers.get_output(out)

loss = lasagne.objectives.squared_error(prediction, target_var)
params = lasagne.layers.get_all_params(out, trainable=True)
updates = lasagne.updates.adam(loss, params, learning_rate=0.0001)
train_fn = theano.function([input_var,target_var],[loss,prediction])

#would then use the following to do one training update, where "inputs" and "targets"
#are your training data:
trn_loss,trn_pred = train_fn(inputs,targets)
```

Questions?

Backpropagation

Vectorized notation: $a_j = f\left(\sum_k w_{jk} x_k + b_j\right) \rightarrow f(wx+b)$

Layer-by layer: $a^l = f\left(w^l a^{l-1} + b^l\right) = f(z^l)$

a_j j^{th} node activation

b_j j^{th} node bias

w_{jk} j^{th} node in layer l, k^{th} node in $l-1$

f applied element-wise

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}$$

$$\delta_j^{N_l} = \frac{\partial C}{\partial a_j^{N_l}} f'(z_j^{N_l}) \rightarrow \delta^{N_l} = \nabla_a C \odot f'(z^{N_l})$$

$$\delta_j^l = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l}$$

$$= \sum_k w_{kj}^{l+1} \delta_k^{l+1} f'(z_j^l)$$

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1}$$
$$= \sum_j w_{kj}^{l+1} f(z_j^l) + b_k^{l+1}$$

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} f'(z_j^l)$$

For each training instance:

1. Forward Pass:

For $l = 1, 2, 3 \ldots N_l$

$z^l = w^l a^{l-1} + b$

$a^l = f(z^l)$

2. 'Error':

$\delta^{N_l} = \nabla_a C \odot f'(z^{N_l})$

3. Backward Pass:

For $l = N_l - 1, N_l - 2, \ldots 1$

$\delta^l = w^{l+1} \delta^{l+1} \odot f'(z^l)$

4. Final Derivatives:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \qquad \frac{\partial C}{\partial b_j^l} = \delta_j^l$$